# Associative Memory as a Hashed Linear Readout

A simple way to view associative memory is:

1. **Hash the input** into a high-dimensional binary vector
2. **Apply a linear readout** to produce the output

Formally:

- Input: $x \in \mathbb{R}^d$
- Hash: $h(x) \in \{-1, +1\}^m$
- Output: $y = Wh(x)$, where $W \in \mathbb{R}^{k \times m}$

## 1. Binary Hash as Random Code

If h(x) behaves like a random ±1 code:

- Different inputs map to **nearly orthogonal** vectors in expectation
- The system resembles a **random feature expansion**

Learning n associations $(x_i \rightarrow y_i)$ reduces to solving:

$Wh(x_i) = y_i$

---

# Behavior vs Storage Capacity

Let m be the hash dimension (number of “features”). And considering the weight matrix alone after training (because pure hashing throws away some of the abilities of the weighted sum.)

## Under Capacity (m<n)

- The weight matrix is **overdetermined**
- Solutions exist with redundancy
- Effects:
    - **Exact recall** for stored associations
    - **Error correction**: noise in h(x) is averaged out
    - Output variance is reduced

### At Capacity (m≈n)

- The system becomes **critically conditioned**
- The norm of W grows
- Effects:
    - High sensitivity to perturbations
    - Small input noise → large output changes

### Over Capacity (m>n)

- The system is **underdetermined**
- No exact solution exists
- Effects:
    - Recall contains **Gaussian-like error**
    - Crosstalk between stored patterns dominates

---

# Response to Unseen Inputs

For inputs not in the training set:

- h(x) looks like a **random ±1 vector**
- The output:

  y = Wh(x)

  becomes a **sum of many weakly (or non) correlated terms**

By the central limit effect:

- Output behaves like **Gaussian noise**
- Mean ≈ 0 (if weights are balanced)
- Variance depends on weight norms

---

# Relaxing the Hash: From Random Codes to Structure

The basic binary hash is too rigid. Relaxing it yields more useful behavior that allows the weight matrix to be used more effectively. For example by allowing the weighted sums to do variance reduction when used under capacity.

### 1. Locality-Sensitive Hash (LSH)

- Nearby inputs → similar hash vectors
- Far inputs → uncorrelated hashes

Effect:

- System becomes **locally generalizing**
- Outputs are meaningful within neighborhoods of learned inputs

---

### 2. Random Projection

Replace discrete hash h(x) = sign(Rx)  with:

h(x) = Rx

where R is a random matrix.

Effect:

- Preserves **geometry (distances, angles)** approximately
- Improves interpolation between stored samples
- Reduces brittleness of binary hashing

---

## Interpretation as Associative Memory

This system behaves like a **distributed associative memory**:

- Storage = embedding inputs into a high-dimensional random space
- Recall = linear superposition
- Interference = capacity limit

It connects directly to:

- Random feature models
- Kernel approximations
- Linear readout neural systems

---

## Connections and Examples

- **Extreme Learning Machines (ELMs)**
  Random hidden layer + trained linear output
  → essentially this model with continuous random projections
- Work by James S. Albus
  CMAC (Cerebellar Model Articulation Controller):
    - Uses **coarse coding / hashing**
    - Achieves **local generalization + fast learning**
    - A practical early instance of this idea

---

# Key Insight

The hashing viewpoint makes something explicit:

> Associative memory can be seen as **linear regression over randomized encodings**, where capacity, generalization, and noise sensitivity are governed by the geometry of the encoding.

- Pure random hashes → exact recall + noise behavior
- Locality-sensitive structure → useful generalization
- Random projections → smooth interpolation

---

# Takeaway

What starts as a seemingly crude “bit hash + linear readout” becomes, with small relaxations, a powerful and widely used pattern:

- Random features → linear solve → associative recall
- Capacity ↔ noise ↔ generalization trade-off
- Geometry of the embedding is everything